VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Navigační aplikace

Navigation Application

Adam Śmieja

Bakalářská práce

Vedoucí práce: Ing. Jan Janoušek

Ostrava, 2021

Abstrakt

Práce se zabývá vývojem dvou na sobě závislých aplikací ke správě turistických tras. První aplikace je vyvinuta pro mobilní telefony s operačním systémem Android a druhá aplikace je pro Connect IQ zařízení společnosti Garmin. V teoretické části je popsána platforma Connect IQ spolu s objektově orientovaným jazykem Monkey C a nástroj Flutter, s jehož pomocí je vyvinuta aplikace pro mobilní telefony. V části praktické je popsán návrh obou aplikací.

Klíčová slova

Flutter; Connect IQ; Monkey C; GeoJSON

Abstract

This bachelor thesis deals with development of two interdependent applications used for managing hiking trails. First application is developed for mobile phones with Android operating system. The second one is for Connect IQ platform compatible devices from Garmin. Theoretical part of the thesis describes the Connect IQ platform together with object oriented language Monkey C and Flutter framework, which is used for the development of the mobile phone application. Practical part deals with the design and architecture of both applications.

Keywords

Flutter; Connect IQ; Monkey C; GeoJSON

Poděkování

Chtěl bych poděkovat Ing. Janu Janouškovi za vedení a dohled při vytváření této bakalářské práce, cenné rady a odbornou pomoc.

Obsah

Seznam použitých zkratek a symbolů

CIQ	–	Connect IQ
API	–	Application Programming Interface
UUID	–	Universally Unique Identifier
XML	–	Extensible Markup Language
SDK	–	Software Development Kit
IoT	–	Internet Of Things
VM	–	Virtual Machine
JIT	–	Just In Time
AOT	–	Ahead Of Time
UI	–	User Interface
GPX	–	GPS Exchange Format
BLoC	–	Business Logic Component
ORS	–	OpenRouteService
OSM	–	OpenStreetMap

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Cílem práce je vyvinout dvě spolupracující aplikace, které budou uživateli sloužit ke správě turistických tras a následnému použití pro navigaci v terénu. K vývoji mobilní aplikace bude použit nástroj Flutter společnosti Google. Flutter je poměrně nový nástroj, díky kterému je možné velmi rychle vyvíjet multiplatformní aplikace s nativním výkonem. Aplikace by měla otestovat možnosti Flutteru využívat složitější komponenty, jako např. mapy, nebo komunikaci s dalším zařízením. Druhá aplikace, která se stará o navigaci, bude vytvořena pro sporttestery Garmin. Díky Connect IQ platformě je možné vyvíjet spoustu druhů aplikací na nositelná zařízení Garmin. K vývoji se používá jazyk Monkey C, který není moc známý, ale jeho inspirace známějšími jazyky zaručuje rychlou adaptaci. Některé modely nenabízí možnost navigace pomocí vestavěných aplikací, a proto vznikne aplikace, která se pokusí nahradit tuto funkcionalitu.

V první části práce popisuje platformu Connect IQ, rozdělení aplikací a jazyk Monkey C. Druhá část se věnuje jazyku Dart, jeho základním vlastnostem a knihovnám. Popisuje také nástroj Flutter, zejména jednotlivé vrstvy, ze kterých se skládá a widgety. Třetí část obsahuje popis vyvinuté aplikace na mobilní telefony s pomocí Flutteru. Vysvětluje state management v aplikaci, využití Google Firebase a proces od vytvoření nové trasy až po navigaci pomocí dané trasy. Čtvrtá část popisuje aplikaci vyvinutou na Garmin sporttestery. Vysvětluje jakým způsobem je zajištěna kompatibilita s více druhy sporttesterů, jak funguje navigace a fungování jednotlivých pohledů.

Kapitola 2

Platforma Connect IQ

Connect IQ [1] je platforma vyvinutá společností Garmin pro vývoj a podporu aplikací na jejich sporttestery a doplňky pro sportovce. Platforma byla oznámena 24. září 2014[2] a změnila přístup třetích stran k vývoji aplikací na zařízení značky Garmin. CIQ byla spuštěna v lednu roku 2015. Pomocí této platformy jsou vývojaři schopni pracovat se zařizeními, které využívají rozdílné senzory a funkce. Garmin se při vývoji drží strategie účelového vývinu zařízení. V praxi to znamená, že jednotlivá zařízení se můžou specializovat na určitý druh sportu, nebo sportovního zaměření.

2.1 Typy aplikací

Aplikace je plně funkční software, který běží na hodinkách. Aplikaci lze přirovnat k těm, se kterými se běžně setkáváme na mobilních telefonech. Tento typ tvůrcům umožňuje přístup a využití všech typů senzorů, které dané zařízení nabízí, komunikaci s telefonem pomocí Bluetooth, vytváření FIT souborů a vysokou interaktivitu s uživatelem oproti jiným typům. K hlavním nevýhodám patří zejména vysoká náročnost na baterii, která se odvijí od využití senzorů, frekvencí překreslování displeje apod. Využití aplikací je velmi široké a závisí od toho, pro jaký účel je navržena.

Datové pole si mohou uživatelé přidat do již existujících datových stránek na jejich zařízení, nebo spustit uvnitř podporované aktivity, což nejčastěji bývá aplikace. Může zobrazit mnoho různých informací jako například čas kola, průměrné tempo, vzdálenost, nebo jinou metriku odvozenou z dostupných dat. Tvůrci mají možnost vytvářet nové datové pole a na pozadí implementovat složité algoritmy pro co nejpřesnější výpočty.

Watch Face jsou vzhledy hodinek samotných, které běží v režimu nízké spotřeby. To znamená, že jsou aktualizovány jednou za minutu a na hodinkách jsou spuštěny non-stop, narozdíl od hodinek jiných výrobců. Tento typ má nejvíce omezený přístup k API a nemůže využívat data například z GPS. Nevhodně navržený watch face rapidně snižuje výdrž baterie.

Widgety jsou obrazovky, na které může uživatel nahlédnout a na první pohled zjistit důležité informace. Zvládají základní vstupy od uživatele. Cílem je prezentované informace podávat v co nejsrozumitelnější formě a pochopitelné na první pohled. Výhodou widgetů je možnost komunikace s telefonem pomocí Bluetooth. Příkladem může být přehled počasí, notifikací, nebo aktivit za daný týden.

Aplikace poskytující audio záznam umožňuje synchronizovat audio záznamy třetích stran se zařízením a poslouchat svou oblíbenou hudbu, podcasty a audio knihy bez nutnosti nošení telefonu neustále u sebe.

2.2 Publikování aplikací

Connect IQ dovoluje publikovat vytvořené aplikace do internetového obchodu Connect IQ Store. Před publikovaním by aplikace měla projít fází testování. Testování ulehčuje možnost publikování beta verzí aplikace, které jsou dostupné jen pro jejich autory. K využití této funkčnosti je nutné pro beta verzi vygenerovat odlišné UUID. Aplikace v obchodě mohou být zdarma, placené, nebo poskytovat prémiové funkce za poplatek.

2.2.1 Konfigurační soubor

Všechny Connect IQ aplikace vyžadují konfigurační soubor, kterému se říká *manifest*. Soubor je ve formátu XML a obsahuje potřebné informace k sestavení aplikace. V manifestu lze nalézt základní informace o aplikaci jako např. podporované zařízení a jazyky, požadované opravnění a další. Element *Application* obsahuje základní atributy, které nám popisují vlastnosti aplikace.

- *entry* – název vstupní třídy pro start aplikace, měla by dědit z *Application.AppBase*
- *id* – jedinečné UUID
- *minSdkVersion* – nejnižší podporovaná verze SDK
- *name* – název aplikace, v našem případě odkazuje na řetězec *AppName*, v kterém je název uložen
- *type* – informace o jaký typ aplikace se jedná
- *version* – určuje verzi aplikace

```
<iq:manifest xmlns:iq="http://www.garmin.com/xml/connectiq" version="3">
<iq:application entry="NavApp" id="199253b5157b4c2c93e5833af0af44e1" launcherIcon=
    "@Drawables.LauncherIcon" minSdkVersion="3.0.0" name="@Strings.AppName" type="
    watch-app" version="0.0.1">
    <iq:products>
        <iq:product id="vivoactive4s"/>
    </iq:products>
    <iq:permissions>
        <iq:uses-permission id="BluetoothLowEnergy"/>
        <iq:uses-permission id="Communications"/>
        <iq:uses-permission id="PersistedContent"/>
        <iq:uses-permission id="PersistedLocations"/>
        <iq:uses-permission id="Positioning"/>
    </iq:permissions>
    <iq:languages>
        <iq:language>ces</iq:language>
    </iq:languages>
    <iq:barrels/>
</iq:application>
</iq:manifest>
```

Výpis kódu 2.1: Manifest soubor

Jako další se v souboru nachází element *products*, kde najdeme výčet podporovaných zařízení, element *permissions* s výčtem požadovaných oprávnění, element *languages*, kde lze najít podporované jazyky a element *barrels*, pokud naše aplikace nějaký využívá.

2.3 Connect IQ API

Celé Connect IQ API spadá pod jmenný prostor, který se nazývá Toybox. Toybox je dále rozdělen na moduly, kde každý z modulů se stará o konkrétní funkcionalitu API[3].

Application je modul, který obsahuje základní třídu každé Connect IQ aplikace. Třída *AppBase* zodpovídá za životní cyklus celé aplikace. Jako další je zde také modul pro přístup k vlastnostem aplikace *Properties* a modul *Storage* k práci s interním uložištěm zařízení.

Modul WatchUI má na starost uživatelské rozhraní a komunikaci s uživatelem. Modul obsahuje třídy k reprezentaci pohledů(*View*, *GlanceView*, *MapView*), ovládání, zobrazování menu a přijímání vstupu od uživatele.

Modul Communications obstarává komunikaci s mobilním telefonem pomocí Bluetooth Low Energy[4]. Mobilní telefon může data sdílet jak se zařízením, tak i vystupovat v roli mostu mezi aplikací a internetem. Tohle umožňuje ze zařízení udělat součást IoT.

Modul Position poskytuje rozhraní k pozičním senzorům na zařízení. Díky *Position* modulu může programátor získat informace o aktuální poloze zařízení, určovat zdroj dat a získávat informace o kvalitě signálu.

Tabulka 2.1: Přehled podporovaných API dle typu aplikace[5]

Název modulu	Datové pole	Watch Face	Widget	Aplikace
Activity	•			•
ActivityMonitor	•	•	•	•
Ant	•		•	•
Application	•	•	•	•
Attention	•		•	•
ActivityRecording				•
Communications			•	•
FitContributor	•			•
Graphics	•	•	•	•
Lang	•	•	•	•
Math	•	•	•	•
PersistedLocations				•
Positioning			•	•
Sensor			•	•
Sensor History	•	•	•	•
System	•	•	•	•
Time	•	•	•	•
Timer		•	•	•
UserProfile	•	•	•	•
WatchUI	•	•	•	•

2.3.1 Životní cyklus aplikace

Aplikace implementuje tři fáze – start, získání prvního pohledu a ukončení. Potom co je aplikace načtena do paměti, je vytvořen objekt aplikace, který je k dispozici po celou dobu běhu a lze ho získat voláním *Application.getApp()*. Po vytvoření objektu aplikace se volá *onStart(state)* metoda,

kde by mělo dojít k inicializaci a obnovení stavu. Následně je požadován první pohled aplikace, který se liší dle funkcionalit.

- *getInitialView()* – primární metoda pro start, vrací základní pohled aplikace, widgetu, watch face, nebo datového pole
- *getGlanceView()* – pokud aplikace implementuje *GlanceView*, zavolá se tato metoda a vrátí list se všemi dostupnými *Glance*[1]
- *getGoalView()* – používá se, pokud aplikace přepisuje základní pohled cíle
- *getPlaybackConfigurationView()* – jedná-li se o audio aplikaci, je potřeba implementovat tento pohled s ovládáním přehrávače

Při ukončení aplikace se volá funkce *onStop(state)*. V tento moment je vhodné uložit stav aplikace do paměti zařízení.

2.4 Monkey C

Monkey C[6] je objektově orientovaný jazyk vyvíjený společností Garmin speciálně pro jejich zařízení. Jazyk se inspiroval známějšími jazyky jako je C[7], Java[8], JavaScript[9], Python[10], Lua[11], Ruby[12] a PHP[13]. To znamená, že pokud se vývojář setkal s některým z těchto známějších jazyků, je pro něj velmi jednoduché začít pracovat s Monkey C.

2.4.1 Rozdíly mezi jazyky

- Java
  - Monkey C je stejně jako Java překládán do byte kódu a dále interpretován pomocí virtuálního stroje
  - Všechny objekty jsou alokovány na haldě a VM se stará o správu paměti. Java pomocí garbage collectoru, Monkey C používá počítání referencí
  - Narozdíl od Javy Monkey C nepoužívá primitivní typy – integer, float a char jsou objekty
  - Monkey C je „duck typed“[14]
  - Monkey C kompilátor nekontroluje typy, místo toho vyhodí chybu za běhu
- Lua/JavaScript
  - Narozdíl od JavaScriptu a Lua v Monkey C nelze předávat metody jako zpětné volání, protože jsou vázány na objekt, v kterém byly vytvořeny

[1] Omezený prostor na displeji

2.4.2 Počítání referencí

Monkey C používá ke správě paměti počítání referencí. To znamená, že objekty jsou z paměti odstraněny až v ten moment, kdy počet referencí na ně se rovná 0. Díky tomu je možné dosáhnout rychlého uvolňování paměti, které je v prostředích s omezeným množstvím paměti důležité. Problém nastává v případě, kdy dojde ke kruhové referenci – objekty odkazují na sebe navzájem. Důsledkem tohoto jevu je, že ikdyž se objekty již nepoužívají, k jejich uvolnění nedojde, protože počet referencí se nerovná 0. V praxi je někdy důležité, aby objekty na sebe navzájem odkazovaly, proto se používá tzv. slabá reference. Slabá reference nenavyšuje počet referencí na objekt a díky tomu lze objekt z paměti odstranit. Takové případy je nutno ošetřit.

2.4.3 Symboly

Jsou konstantní identifikátory. Pokud kompilátor narazí na nový symbol, automaticky mu přiřadí novou unikátní hodnotu. To umožňuje využívat symboly jako klíče ve slovnících, nebo konstanty bez deklarace explicitních enumerátorů.

```
using Toybox.System;

var a = :symbol_1;
var b = :symbol_1;
var c = :symbol_2;
System.println(a == b); // Prints true
System.println(a == c); // Prints false
```

Výpis kódu 2.2: Příklad symbolů[6]

2.4.4 Operátory Instanceof a Has

Díky tomu, že je Monkey C „duck typed“ má programátor možnost vysoké flexibility ve psaní kódu. Problémem je, že kompilátor nemůže provést kontrolu typů. *instanceof* a *has* nám umožňuje kontrolovat typy za běhu aplikace. Operátor *instanceof* kontroluje, zda daný objekt dědí z určité třídy. Operátor *has* kontroluje, zda daný objekt obsahuje symbol, kterým může být veřejná metoda, instanční proměnná, nebo i definice třídy a modul.

```
using Toybox.System;
using Toybox.Sensor as Sensor;

var value = 5;
if (value instanceof Lang.Number) {
    System.println("Value is a number");
}

var sensorInfo = Sensor.getInfo();
if (sensorInfo has :accel && sensorInfo.accel != null) {
    var accel = sensorInfo.accel;
    var xAccel = accel[0];
    var yAccel = accel[1];
    System.println("x: " + xAccel + ", y: " + yAccel);
}
```

Výpis kódu 2.3: Použití operátorů instanceof a has[6]

Kapitola 3

Flutter

Flutter je multiplatformní nástroj vyvíjený společností Google. K jeho představení došlo v roce 2015 a o tři roky později, 4. prosince 2018 byla uvedena jeho první stabilní verze. Je napsaný pomocí C[7], C++[15] a Dartu[16]. „Out of the box“ Flutter vývojářům nabízí základní ovládácí prvky uživatelského rozhraní v designovém jazyku Androidu(Material design) a iOS(Cupertino). Mezi hlavní výhody Flutteru můžeme zařadit jeho multiplatformnost, rychlost vývoje díky hot-reloadu a podporu ze strany Googlu. Aplikace jako např. Google Ads, Stadia, Baidu, The New York Times a další používají Flutter[17].

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(title: Text('Welcome to Flutter')),
        body: Center(child: Text('Hello World')),
      ),
    );
  }
}
```

Výpis kódu 3.1: Příklad Hello World aplikace ve Flutteru

3.1 Dart

Je programovací jazyk vyvíjený stejně jako Flutter Googlem. Je to objektově orientovaný jazyk, který ke správě paměti využívá garbage collector a podporuje rozhraní, mixiny, abstraktní třídy, generické typy a odvození typů. Dart se kompiluje buď do nativního kódu, a nebo pomocí dart2js kompilátoru do JavaScriptu.

3.1.1 Typový systém Dartu

Dart je typově bezpečný, používá kombinaci statické kontroly typů a kontroly za běhu aplikace k zajištění toho, aby hodnota proměnné vždy odpovídala jejímu statickému typu. Deklarování typů není povinné díky odvození typů. V případech, kdy je potřebný dynamický kód lze použít i typ *dynamic*, u kterého se kontrola provádí až za běhu aplikace.[18]

3.1.2 Dart kompilátor

Kompilátor nám umožňuje spustit kód na dvou platformách. Pro nativní platformy má Dart virtuální stroj, který poskytuje Just-In-Time i Ahead-Of-Time kompilace pro vytvoření strojového kódu.

- JIT - používá se zejména v průběhu vývoje aplikace, kdy dochází k častým změnám kódu
- AOT - celý kód se zkompiluje ještě před samotným spuštěním aplikace a je pak rychlejší

Pro webovou platformu Dart obsahuje dva kompilátory. Dartdevc je kompilátor, který se využívá při vývoji a dart2js pro produkční kód. Oba dva překladájí Dart do JavaScriptu[19].

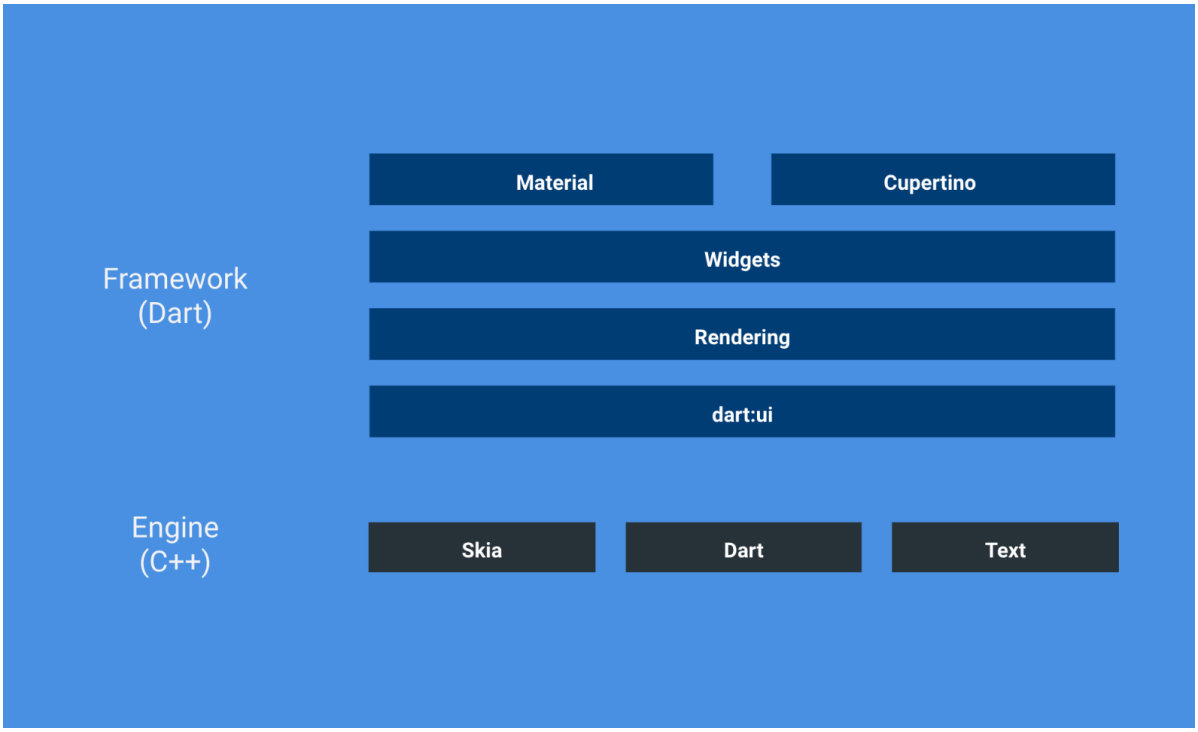
3.1.3 Dart knihovny

Knihovny dále rozšiřují funkcionalitu jazyka. Mezi ty nejdůležitější patří **dart:core**, která obsahuje vestavěné typy, kolekce vyjímky atd., rozšířenější kolekce lze najít v knihovně **dart:collection**. K matematickým operacím a generování náhodných hodnot je určena knihovna **dart:math**. Asynchronní programování v Dartu je realizováno pomocí knihovny **dart:async**, která implementuje Future a Streamy. Souběžné provádění kódu se provádí za pomocí izolátů. Izoláty se podobají vláknům z jiných jazyků(C#, Java), ale nesdílí mezi sebou paměť a komunikují spolu pouze pomocí zpráv.

3.2 Architektura Flutteru

Flutter poskytuje moderní, reaktivní rozhraní psané v Dartu. Skládá se ze 4 vrstev. Dá se říci, že s čím vyšší vrstvou zrovna pracujeme, o to lehčí je práce s ní, ale poskytuje méně možností pro

přízpůsobení a naopak, s čím nižší vrstvou pracujeme, o to složitější je práce, ale dovoluje nám mnohem větší možnosti přízpůsobení.



Obrázek 3.1: Vrstvy Flutteru[20]

3.2.1 dart:ui

Vrstva poskytuje nejzákladnější funkce nástroje. Komunikuje přímo s vykreslovacím Skia enginem a dává nám přístup ke třídám jako např. *Canvas*, *Paint* a *TextBox*. Pokud bychom pracovali pouze s touto vrstvou, tak jsme nuceni ručně vykreslit všechny prvky našeho UI pro každý snímek a implementovat a spravovat veškerou interakci s uživatelem.

3.2.2 Vykreslovací knihovna

Vykreslovací knihovna je první abstrakční vrstvou nad **dart:ui** knihovnou a stará se za nás o všechny výpočty nad UI. K tomu používá *RenderObject*. Z těchto objektů poté sestaví strom a Flutter jej vykreslí na obrazovku. K optimizaci tohoto procesu Flutter používá algoritmus, při kterém si ukládá výsledky operací. Většinu času Flutter k sestavení UI používá *RenderBox*. Výsledkem je box-layout, který umožňuje překreslení jen potřebných částí.

3.2.3 Knihovna widgetů

Je další vrstvou abstrakce, která poskytuje základní widgety, které můžeme použít v naší aplikaci. Všechny widgety spadají do jedné ze tří kategorií:

- **Layout** - např. *Column* a *Row* - usnadňují rozložení widgetů
- **Painting** - např. *Text* a *Image* - umožňují nám vykreslit obsah na obrazovku
- **Hit-Testing** - např. *GestureDetector* - umožňuje nám zaregistrovat různé vstupy od uživatele(dotyky, táhnutí apod.)

Většinu času se pracuje právě s touto knihovnou a pomocí ní se pak skládají vlastní, složitější widgety.

3.2.4 Material a Cupertino knihovny

Čtvrtá vrstva se skládá z předpřipravených widgetů, které můžeme rovnou použít. Widgety se řídí designovým jazykem platforem, pro kterou jsou určeny. Snaží se kopírovat vzhled a chování nativních prvků.

3.2.5 Widget

Widget je základním stavebním blokem uživatelského rozhraní. Widgety tvoří hierarchii, která je založená na jejich kompozici. Každý widget je vnořen uvnitř svého rodiče, od kterého může obdržet jeho kontext. Typicky jsou widgety složeny z mnoha dalších, menších jednoúčelových widgetů a dohromady tvoří komplexní bloky. Flutter používá widgety k reprezentaci uživatelského rozhraní, interakce s uživatelem, řízení stavu aplikace, animací a navigace. K dispozici jsou dva druhy widgetů: stateless a stateful[21].

3.2.6 State

Widgety jsou samy o sobě neměnné, a proto k implementaci stavu je nutno použít třídu *State*, pomocí které vyjadřujeme vnitřní stav widgetu. Vnitřním stavem můžeme chápat jeho vzhled, funkčnost, nebo hodnoty proměnných. Pokud dojde ke změně stavu, je na každém widgetu, aby korektně implementoval upozornění na změnu stavu.

3.2.7 StatelessWidget

Nemají proměnlivý stav, to znamená, že nemají žádné vlastnosti, které by se v průběhu jeho životního cyklu mohly změnit. Často se jedná o widget, který v sobě zahrnuje další widgety, ty reprezentují stav. Správně se *build()* metoda StatelessWidgetu volá jen při jeho vložení do stromu widgetů. Pokud se volá přílíš často, např. při změně konfigurace potomků, je pak doporučené zvolit *StatefulWidget*, který je pro toto chování přizpůsoben.

3.2.8 StatefulWidget

Zpravidla obsahují vlastnosti, které se v čase mění a je potřeba tento stav reflektovat v uživatelském rozhraní. K tomu je potřeba widget znovu sestavit pomocí *build()* metody. *StatefulWidget* sám o sobě nemá *build()* metodu, místo toho je jeho rozhraní sestaveno pomocí State objektu. Kdykoliv se změní State objekt, je potřeba volat metodu *setState()*, která signalizuje potřebu znovu sestavit část uživatelského rozhraní pomocí *build()* metody State objektu.

Ostatní widgety můžou díky oddělení stavu od widgetu zacházet se stateless i stateful widgety stejným způsobem. Místo nutnosti uložení potomků k zachování stavu je rodič schopen kdykoliv vytvořit novou instanci potomka, aniž by ztratil jeho stav. Flutter se pak postará o nalezení a znovupoužití posledního stavu.

Kapitola 4

Návrh a implementace mobilní aplikace

Kapitola se zabývá vývojem, popisem mobilní aplikace a použitých technologií. K prvotnímu přístupu do aplikace je nutné internetové připojení a vytvoření účtu, nebo přihlášení se pomocí účtu Google.

4.1 Analýza zadání

Ze zadání plyne, že je nutné navrhnout a implementovat mobilní aplikaci, která má možnost vytvoření vlastního účtu, na který budou dále vázány vytvořené turistické trasy. Trasy by mělo být možno v aplikaci vytvořit, editovat, smazat a uložit do zařízení pro použití offline. O nalezení trasy se bude starat navigační API třetích stran. Další z možností vytvoření je importování trasy z GPX souboru. Uživatel má možnost přizpůsobení si vyhledání trasy dle nabízených možností - chůze, turistika, cyklistika, nebo horská cyklistika. Odesláním trasy do sporttesteru se zahájí navigace. K dispozici je možnost vytvořit si offline účet, který je vázán na zařízení a slouží k použití předem uložených tras i v případě, že není dostupné připojení k internetu. Výsledná funkcionalita aplikace jde shrnout do dvou částí, první část obsahuje vytváření, editaci, mazání a správu tras a druhou částí je použití tras k navigaci.

4.2 State management v aplikaci

Velmi důležitou roli při vývoji hraje volba správného state management balíčku. Mezi ty nejznámější patří InheritedWidget, BLoC[22], Redux[23], Provider a Riverpod[24]. Zvolil jsem Riverpod, který není závislý na Flutter SDK, je bezpečný při kompilaci a také je dobře testovatelný. Nejdůležitější komponentou jsou providery. Provider můžeme chápat jako přístupový bod v aplikaci, který nám poskytuje aktuální stav. V aplikaci používám dva druhy providerů - *Provider* a *ChangeNotifierProvider*. Pro správnou funkčnost je nutné obalit celou aplikaci pomocí *ProviderScope*.

4.2.1 Providery v aplikaci

Ke správě stavu aplikace používá celkem 4 providery.

- *authServiceProvider* - jedná se o *ChangeNotifierProvider*, spravuje aktuální stav uživatele
- *firestoreProvider* - jedná se o *Provider*, poskytuje proudy aktualizací dat aktuálního uživatele a možnost zápisu do databáze
- *openRouteServiceProvider* - jedná se o *ChangeNotifierProvider*, stará se o komunikaci s OpenRouteService[25] API pro vyhledávání tras
- *watchConnectionProvider* - jedná se o *ChangeNotifierProvider*, hlavním úkolem provideru je komunikace s Garmin zařízením

```
final authServiceProvider = ChangeNotifierProvider<AuthService>(
    (ref) => AuthService.instance(ref.read));
final openRouteServiceProvider = ChangeNotifierProvider<OpenRouteService>(
    (ref) => OpenRouteService.instance(ref.read));
final firestoreProvider =
    Provider<FirestoreService>((ref) => FirestoreService.instance());
final watchConnectionProvider =
    ChangeNotifierProvider<WatchService>((ref) => WatchService());
```

Výpis kódu 4.1: Použité providery v aplikaci

4.2.2 WidgetView vzor

Flutter dokáže pomocí Dartu v jedné třídě mixovat prvky deklarativního i imperativního stylu programování. To způsobuje návaznost mezi uživatelským rozhraním a funkční částí widgetu. Tenhle problém lze řešit třemi způsoby:

1. Rozdělení widgetu do více malých widgetů
    - **Výhody** - dobrá volba, pokud se jedná o widget, který lze použít na více místech aplikace
    - **Nevýhody** - hodně kódu navíc, problémy s předáváním parametrů
2. Použití metod pro sestávení částí widgetu
    - **Výhody** - čitelnost hlavní *build* metody
    - **Nevýhody** - problémy s předáváním parametrů, roztahání kódu na více míst

3. Použití balíčků k izolování stavu od widgetu

    - **Výhody** - dobrá škálovatelnost
    - **Nevýhody** - často až příliš složité pro vyřešení jednoduchého problému

Pro větší widgety jsem se rozhodl použít *WidgetView*[26] vzor, který dobře odděluje uživatelské rozhraní od jeho funkcionality. Myšlenku za tímto vzorem lze aplikovat na stateless i stateful widgety.

```
class MyWidget extends StatefulWidget {
  @override
  _MyWidgetController createState() => _MyWidgetController();
}

class _MyWidgetController extends State<MyWidget> {
  @override
  Widget build(BuildContext context) => _MyWidgetView(this);
}

class _MyWidgetView extends WidgetView<MyWidget, _MyWidgetController> {
  _MyWidgetView(_MyWidgetController state) : super(state);

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

Výpis kódu 4.2: Widget vytvořený pomocí *WidgetView*

Aplikováním tohoto vzoru jsem tedy schopen jasně oddělit vzhled od funkčnosti. Další z výhod je možnost snadného přidání responzivity, kde podle zařízení vykreslíme jiné *View*, ale controller zůstává stejný.

4.3 Autorizace uživatelů a uchování dat online

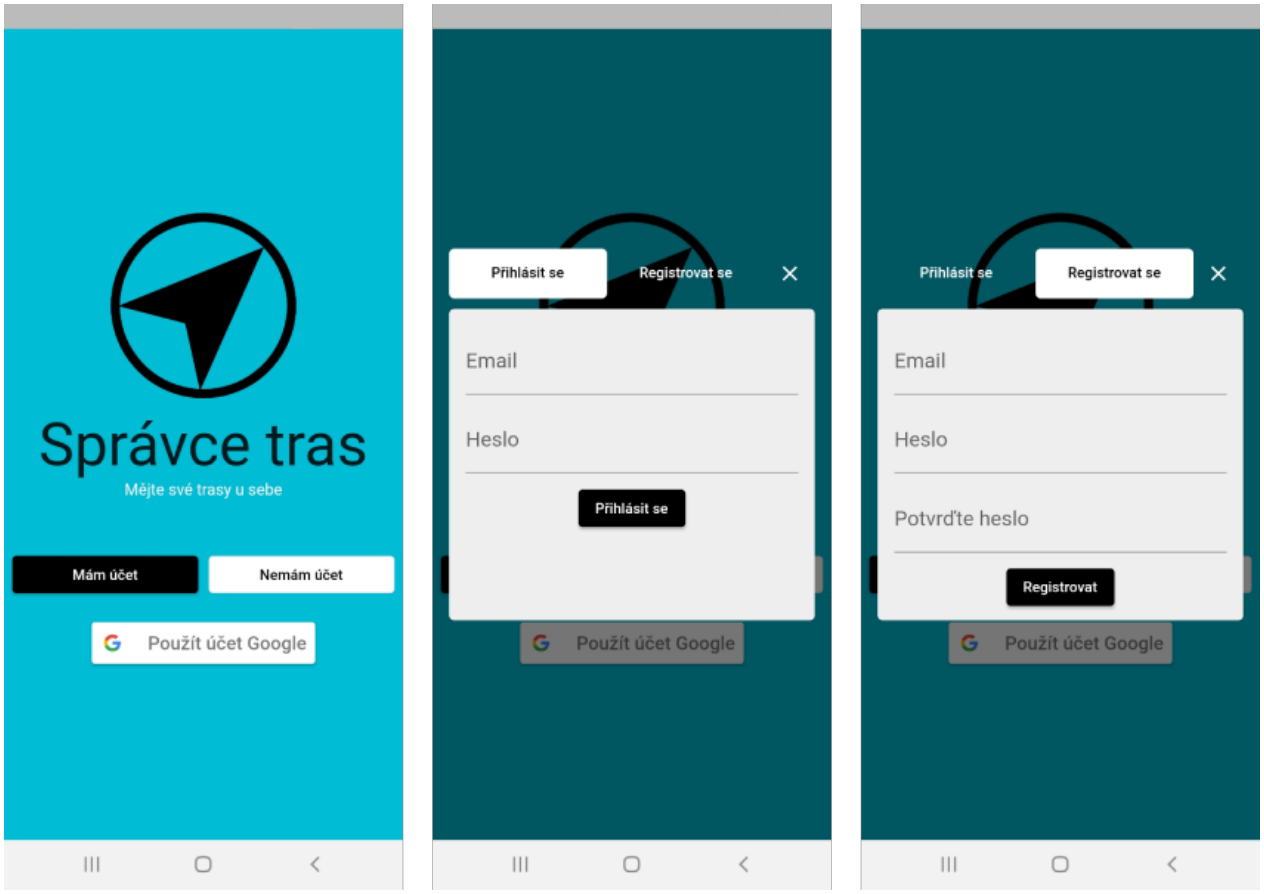
Jedním z požadavků byla autorizace uživatelů a možnost vytvoření si vlastního účtu, který není vázán na zařízení a uživatel tak neztratí data např. při výměně, nebo ztrátě telefonu. Vzhledem k tomu, že cílem práce nebylo navržení takového systému, zvolil jsem mobilní platformu Firebase. Firebase je cloudová služba společnosti Google, která poskytuje autorizaci uživatelů, dva druhy

databáze, web hosting a mnoho dalších funkcí. Využil jsem moduly Firebase Authentication, který se stará o správu a autorizaci uživatelů a modul Cloud Firestore. Cloud Firestore je škálovatelná NoSQL databáze, která běží v cloudu.

4.3.1 Autorizace pomocí Firebase

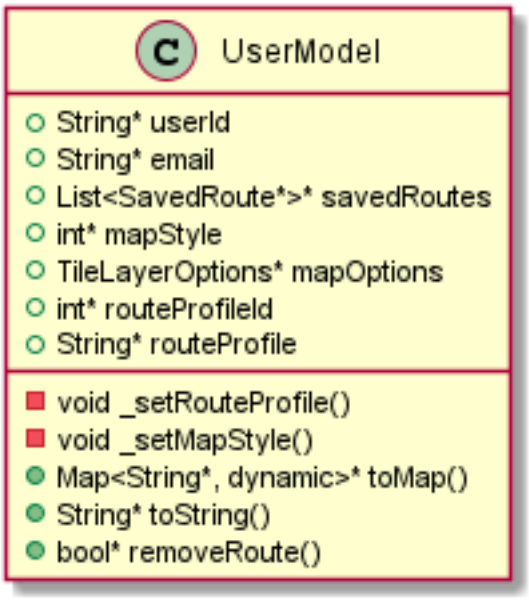
Firebase Authentication poskytuje služby, SDK a knihovny uživatelského rozhraní k autorizaci uživatelů do aplikace. K autorizaci je možné použít vytvořený účet, telefonní číslo, sociální sítě, nebo Apple ID[27] a Google účet. Využívá přitom standardů jako OAuth 2.0[28] a OpenID Connect[29].

V mé aplikaci jsem využil možnost přihlášení pomocí vytvoření účtu s emailem a heslem, nebo přes Google účet z důvodu zpřístupnění aplikace pro více uživatelů. Použitá možnost nemá vliv na další funkčnost aplikace.



Obrázek 4.1: Přihlašovací obrazovka

Třída reprezentující uživatele se jmenuje *UserModel*. Pomocí mixinu *ChangeNotifier* můžeme signalizovat změnu stavu aktuálního uživatele. Obsahuje unikátní identifikátor uživatele, email, seznam uložených tras a informaci o preferovaném stylu mapových podkladů a vyhledávače tras.



Obrázek 4.2: Třídní diagram *UserModel*

4.3.2 Uchování dat v Cloud Firestore

Cloud Firestore je NoSQL databáze. To znamená, že data uchovává ve formě dokumentů a kolekcí. Dokument je základní jednotkou používánou k uchování dat. Data v dokumentu jsou uchována v mapách. Kolekce jsou pak využívány pro sdružování dokumentů do celků. Kolekce jako taková nemůže uchovávat data, pouze dokumenty. Každý dokument musí mít svoje jedinečné ID. Celá databáze nemá jednotné schéma, takže dokumenty v jedné kolekci nemusí obsahovat stejná pole.

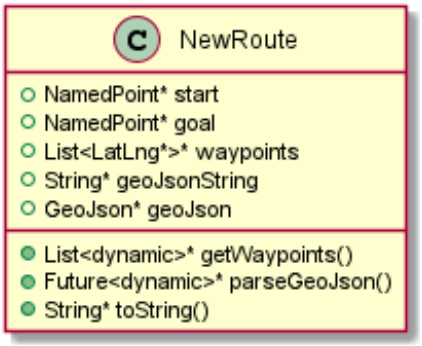
Datový model je jednoduchý. Skládá se z jedné kolekce *users*, která obsahuje data všech uživatelů v podobě dokumentů. U uživatele ukládám jeho ID, emailovou adresu, datum prvního přihlášení, informaci o preferovaném mapovém podkladu a profilu trasy. Každý dokument uživatele obsahuje kolekci *routes*. Tato kolekce seskupuje všechny trasy uživatele do jednotlivých dokumentů.

4.4 Tvorba nové trasy

K tvorbě nové trasy se uživatel dostane z hlavního menu vybráním karty „Nová trasa“. Po načtení se mapa zaměří na uživatelovu aktuální pozici a zvýrazní jí ikonou na mapě. Vytvořit trasu je možné celkem třemi způsoby: zadáním startu a cíle do příslušných textových polí, které se zobrazí kliknutím na ikonku lupy a následným vybráním lokace z dostupného našeptávače, ručním zadáním startu a cíle pomocí kliknutí na mapu, nebo importováním z GPX souboru, který je uložený v

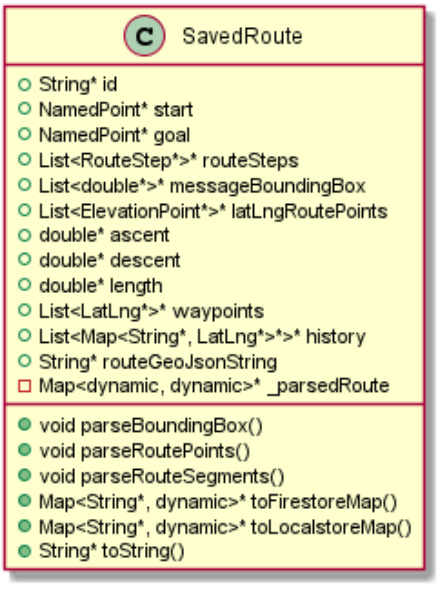
zařízení. Trasu je dále možno upravit vybráním dalších záchytných bodů, přes které má trasa vést, počet takových bodů není nijak omezen.

Při tvorbě trasy aplikace pracuje s objektem *NewRoute*. Tento objekt obsahuje základní informace jako: začátek a konec trasy, seznam záchytných bodů, GeoJSON objekt a GeoJSON převedený do řetězce.



Obrázek 4.3: Třídní diagram *NewRoute*

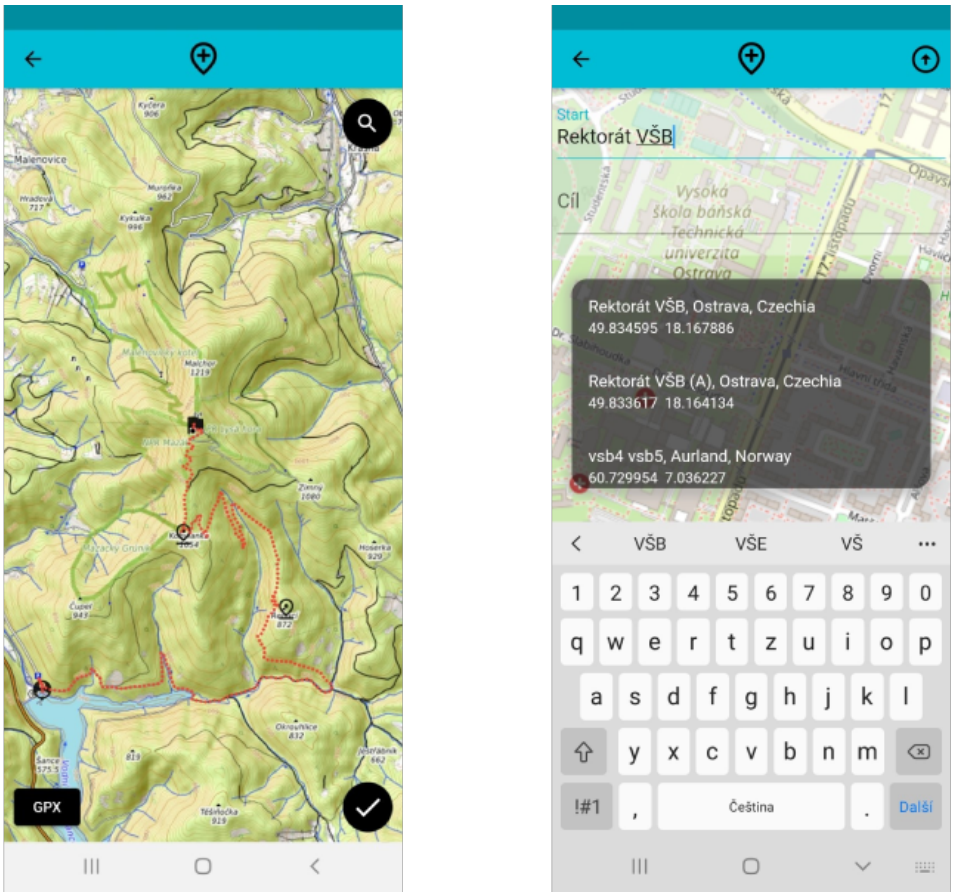
Vytvořené trasy se ukládají jako *SavedRoute* objekty. Oproti *NewRoute* obsahuje navíc data z API jako: list instrukcí, souřadnice ohraničující celou trasu, jednotlivé body pro vykreslení trasy spolu s nadmořskou výškou, celkový sestup a výstup a GeoJSON.



Obrázek 4.4: Třídní diagram *SavedRoute*

4.4.1 Mapový widget

Jako mapový widget jsem použil *flutter_map*[30]. Jedná se o Dart implementaci známého JavaScriptového balíčku Leaflet[31]. Výhodou tohoto balíčku je možnost zvolit si zdroj map. V aplikaci používám open-source data OpenStreetMap[32], OpenTopoMap[33] a WMFLabs[34]. Obsah mapy je rozdělen do vrstev, tzn. že mapový podklad, body na mapě a trasy jsou na svých, na sobě nezávislých vrstvách. K dispozici je *MapController*, který poskytuje základní funkcionalitu. Pro mé potřeby využívám balíček *map_controller*[35], který nabízí *StatefulMapController*. Tento controller obsahuje řadu užitečných funkcí. Důležitou vlastností, proč jsem se rozhodl použít právě tento balíček je funkce *toGeoJson()*, která umožňuje exportovat obsah mapy do GeoJSON souboru.



Obrázek 4.5: Vytvořená trasa(vlevo) a našeptávač(vpravo)

4.4.2 GeoJSON

Jedná se o standard pro zapisování geografických dat, který vychází z velice rozšířeného formátu JSON. Korektně zapsaný GeoJSON je vždy objektem skládajícím se z „name : value“ párů. Do GeoJSONu lze uložit i instrukce pro navigaci. ORS ukládá instrukce do jednotlivých segmentů

trasy. Instrukce se skládá ze vzdálenosti(distance), předpokládaného časového intervalu(duration), typu(type), názvu(name), slovního popisu(instruction) a intervalu bodů, které obsahuje(way_points).

```
{
    "distance": 38.3,
    "duration": 27.6,
    "type": 11,
    "instruction": "Head south",
    "name": "-",
    "way_points": [0, 3]
}
```

Výpis kódu 4.3: Příklad instrukce uložené v GeoJSONu

4.4.3 OpenRouteService API

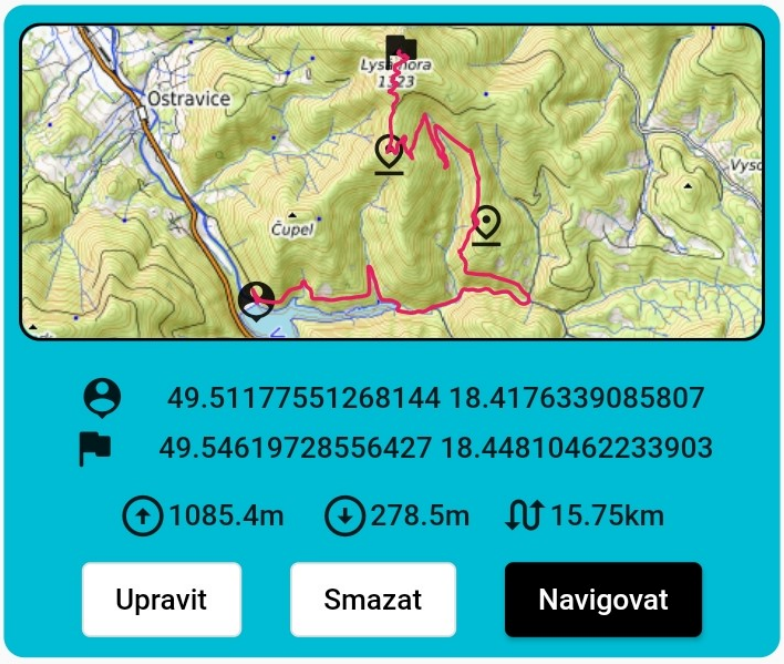
K našeptávání míst a vyhledávání tras aplikace využívá ORS API. Jedná se o open-source prostorovou službu, která využivá OSM dat. Nabízí širokou škálů přizpůsobení, vysoký výkon a je psaná v Javě. K hledání a vytváření tras používá Pelias[36] vyhledávač. K vytvoření trasy aplikace používá */v2/directions/{profile}/geojson* endpoint. {profile} parametr určuje uživatelovu aktuální preferenci vyhledávání. K zaslání požadavku se používá POST metoda a je nutné přidat API klíč, který je v základní verzi zdarma s určitými omezeními. Tělo dotazu tvoří JSON objekt.

4.5 Uložené trasy

Jak je zmíněno výše, trasy se vážou na uživatelský účet. Seznam všech dostupných tras je uživateli k dispozici v menu „Moje trasy“. Jednotlivé trasy jsou zde reprezentovány jako widgety. Každý widget obsahuje informaci o začátku a cíli trasy. Zapsání se liší podle toho, jak uživatel trasu vytvořil. Použil-li našeptávač, uvidí název bodu. Pokud bod vybral dotykem na mapě, tak se zobrazí GPS souřadnice. Dále zde lze najít celkové klesání a stoupání trasy a celkovou délku. Posledním elementem karty jsou tlačítka, která umožňují danou trasu upravit, smazat, a nebo použít k navigaci. Úprava trasy probíhá ve stejném prostředí jako vytvoření nové trasy, s tím rozdílem, že trasa je zde již načtena.

4.5.1 Widget trasy

Jak je vidět na obrázku 4.6, tak widget lze rozdělit do čtyř hlavních částí: mapy, začátku a cíle, informací o trase a tlačítek. Z toho vychází i *build()* metoda widgetu, která sestavuje widget čtyřmi pomocnými funkcemi z důvodu čitelnosti kódu.



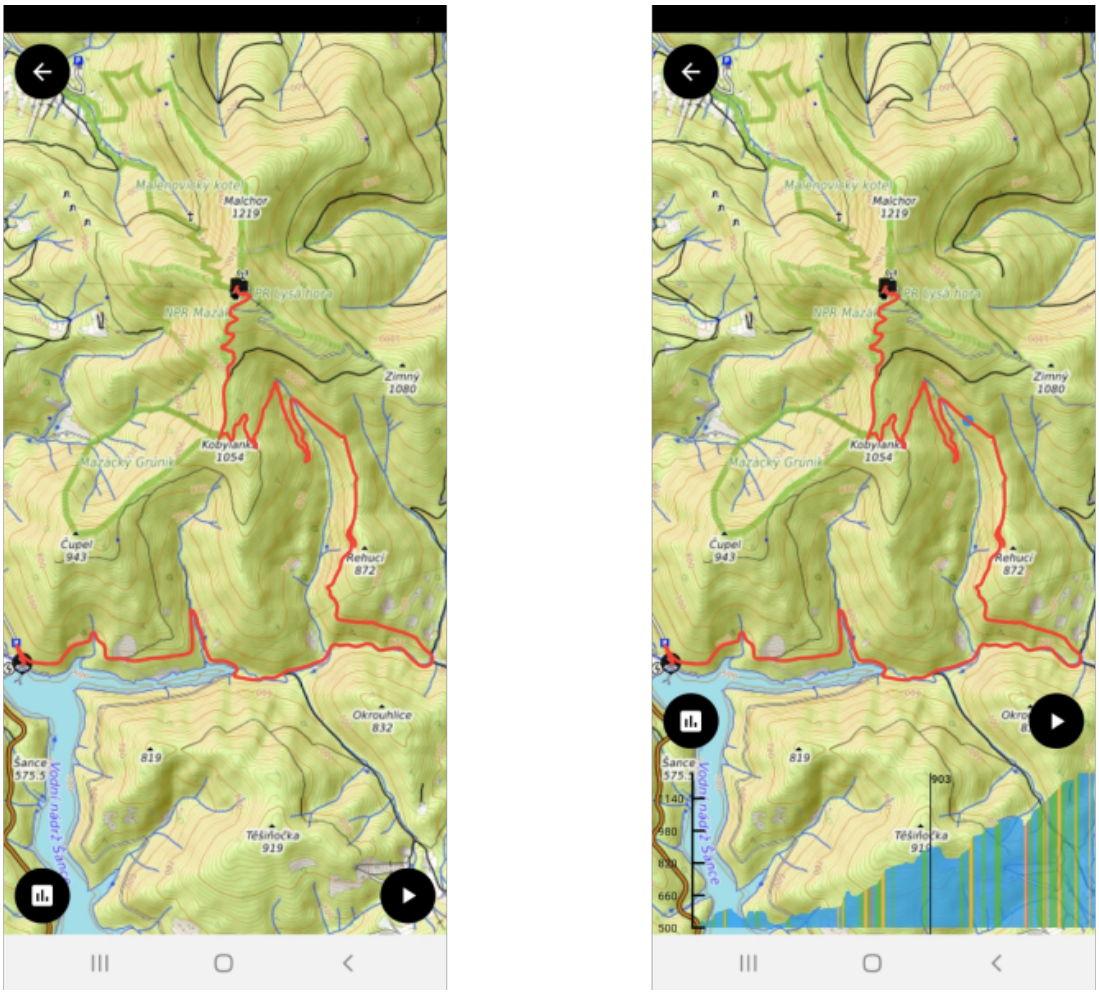
Obrázek 4.6: Widget trasy

Jelikož jsou všechny widgety trasy uspořádané do *SliverList* widgetu, je nutné je opatřit klíči. Bez klíčů dochází k záměně stavů jednotlivých widgetů při opětovném sestavení listu po smazání trasy. K vyřešení tohodle nežádoucího chování jsem použil *UniqueKey*, který každému widgetu přiřadí unikátní klíč, podle kterého lze pak jednoznačně spojit stav a widget.

4.6 Navigace pomocí trasy

Navigace probíhá na speciální obrazovce, na kterou se uživatel dostane po vybrání trasy. Společnost Garmin poskytuje balíček pro komunikaci s jejich zařízením, který je ovšem částečně nefunkční a vývojáři chyby neopravují. Zahájení navigace probíhá v následujícím pořadí:

1. Převedení dat navigace do patřičného formátu
2. Odeslání instrukcí, tím se zahájí přenos dalších částí incializací *EventChannelu*
3. Průběh navigace, telefon průběžně získává informace o aktuální poloze a zobrazuje na mapě

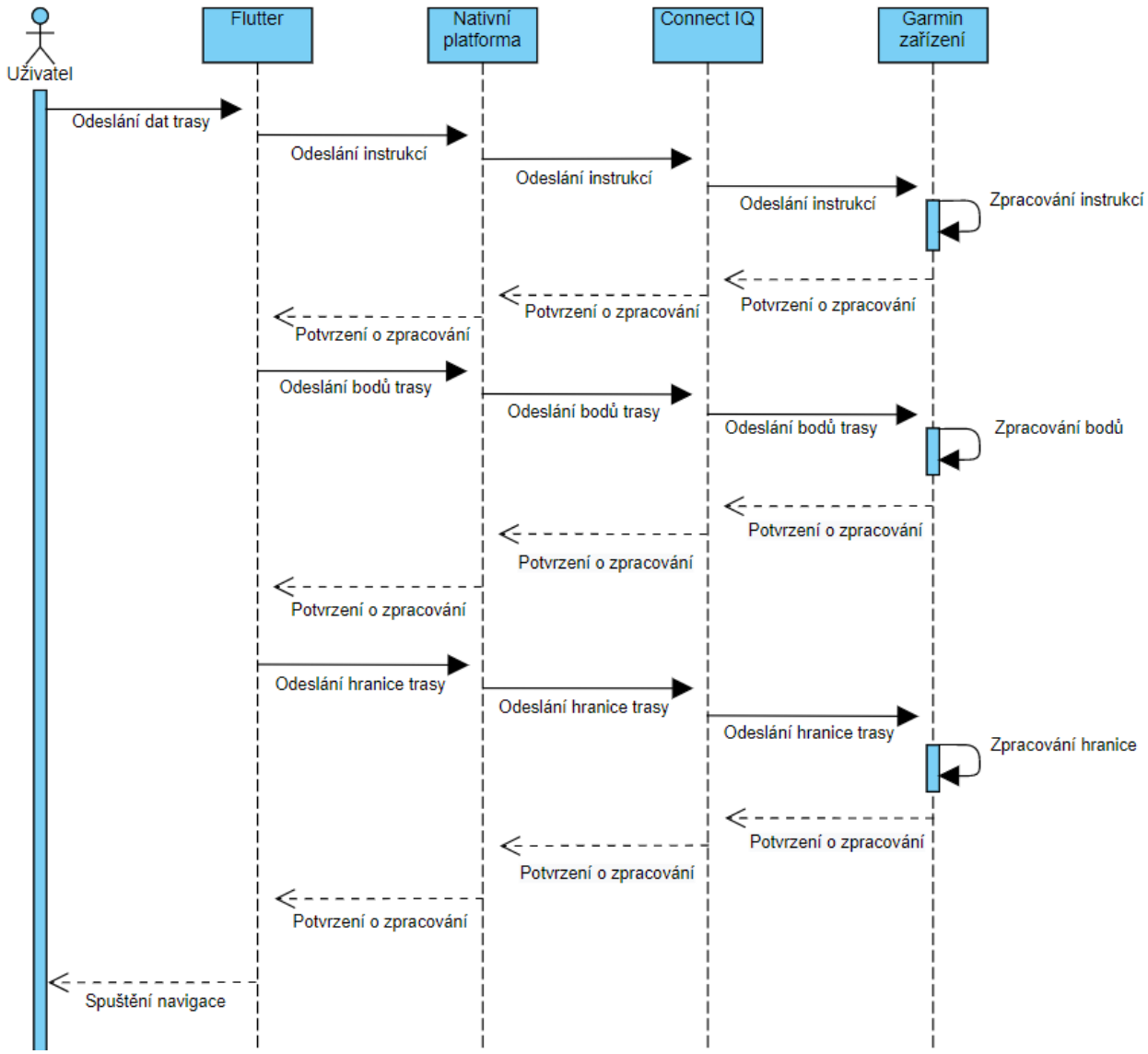


Obrázek 4.7: Obrazovka s navigací

4.6.1 Komunikace s nativním prostředím a Connect IQ

Jelikož se zařízením nejde komunikovat napřímo pomocí Bluetooth a pro Flutter není vytvořený žádný balíček, který by obalil SDK, musel jsem použít SDK poskytované Garminem. K tomuto SDK Garmin přidává ukázkovou aplikaci a dokumentaci. Jak je zmíněno výše, SDK z části nefunguje jak má, a proto je práce s ním obtížná. *MethodChannel* a *EventChannel* jsou dva způsoby komunikace mezi Flutterem a nativní platformou. První zmíněný slouží k asynchronnímu volání metod, druhý ke komunikaci používá proudy, kterým můžeme na straně Flutteru naslouchat a náležitě reagovat. Při komunikaci mezi platformami je možné posílat jen základní typy jako např. int, bool, String, List atd. Chceme-li poslat vlastní třídu, je nutné jí nějakým způsobem serializovat. Stejné omezení platí i pro komunikaci mezi nativní částí a CIQ aplikací. Z tohoto důvodu serializaci dat do map aplikace provádí už při převádění dat do patřičného formátu, aby nebylo nutné je převádět při přenosu mezi Flutterem a nativní platformou a podruhé mezi nativní platformou a CIQ aplikací. Na obrázku 4.8

je ukázán proces posílání dat do Garmin zařízení. Jako první se posílají instrukce k navigaci přes nativní platformu do CIQ aplikace a odtud do Garmin zařízení. Na zařízení se instrukce zpracují a až teprve potom je vyslán zpátky signál informující o výsledku zpracování. V nativní části je pak připraven posluchač událostí, který tyto zprávy dál posílá do připraveného *EventChannelu*. Ten je pak emituje do Flutteru.

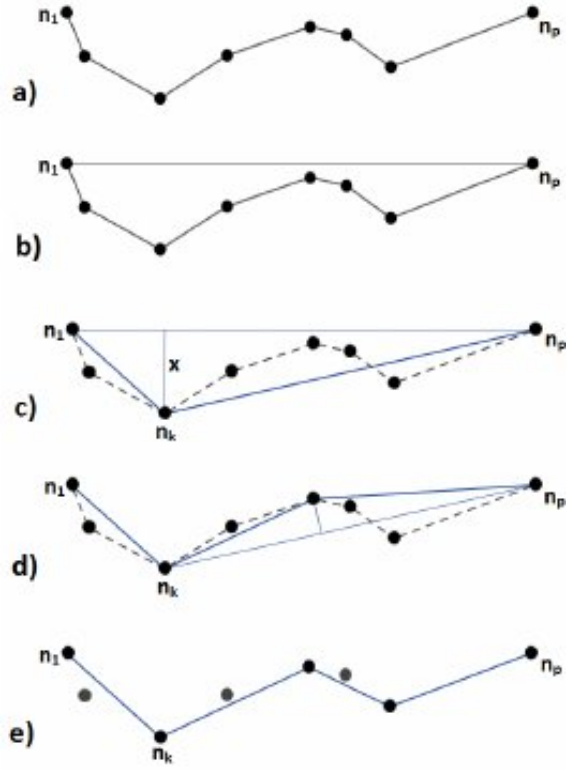


Obrázek 4.8: Komunikace se zařízením Garmin

4.6.2 Optimalizace trasy

Z důvodu omezeného výkonu na Garmin zařízeních je nutné delší trasy optimalizovat. Delší trasy jsou schopny překročit hranici 500 a více bodů. V takovém případě zpracování souřadnic na zařízení trvá moc dlouho a sepne se tzv. „WatchDog“, ve volném překladu hlídací pes, který hlídá vykonávání funkcí a v případě, že vykonání funkce trvá moc dlouho, přeruší jí, aby nedocházelo k zasekávání uživatelského rozhraní. Testováním jsem došel k počtu 250 bodů. V takovém množství je trasa ještě dostatečně detailní a zároveň zařízení nemá problém se zpracováním.

Jako řešení vyvstalého problému jsem použil balíček *simplify_dart*[37]. Jedná se o kombinaci Douglas-Pecker[38] algoritmu a úpravy pomocí radiální vzdálenosti. Cílem algoritmu je optimalizovat trasu a přitom zachovat její přibližnou podobu.



Obrázek 4.9: Průběh optimalizace trasy[39]

Kroky algoritmu jsou následující:

1. Spojí se první a poslední bod trasy
2. Ze zbylých bodů se najde bod, který je nejdál od vzniklé přímky a je v rámci tolerance
3. Rekurzivní volání, dokud se neprojdou všechny body

Kapitola 5

Návrh a implementace Connect IQ aplikace

Kapitola se zabývá vývojem a popisem funkcionalit aplikace pro Garmin sporttestery. Popisuje způsob podpory více druhů zařízení, algoritmus navigace a jednotlivé pohledy aplikace.

5.1 Analýza zadání

Dle zadání je nutné navrhnout a implementovat CIQ aplikaci, která je schopná komunikovat s mobilním telefonem, na kterém je nainstalovaná tzv. „companion“ aplikace. CIQ aplikace musí umět přijmout a zpracovat navigační data. Na základě těchto dat poté uživateli zobrazit aktuální instrukci a počítat celkovou vzdálenost. Aplikace obdrží i seznam všech bodů trasy, ze kterých lze vykreslit přibližný obrys trasy. Na tomto obrysu by uživatel měl vidět svoji aktuální polohu a měl možnost s obrysem pohybovat, přibližovat a oddalovat jako na klasické mapě. Z důvodu přístupnosti pro co nejvíce uživatelů je důležité zajistit kompatibilitu s vysokým počtem zařízení. Aplikace by se měla chovat a vypadat na zařízeních s dotykovým displejem stejně, jako na zařízeních s ovládáním pomocí tlačítek. Stejné omezení platí i pro kruhové a hranaté displeje.

5.2 Podpora více modelů zařízení

Z důvodu zpřístupnění aplikace pro co nejvíce uživatelů jsem musel vyřešit jakým způsobem toho dosáhnout. CIQ API poskytuje více řešení. Prvním řešením je možnost získat číslo modelu pomocí volání *System.getDeviceSettings().partNumber* za běhu aplikace a dle toho uživateli přízpůsobit ovládání a vzhled aplikace. Problémem tohoto řešení je, že čísla modelu se liší dle distribuce. Druhým možným řešením je použít identifikátory dle typu zařízení. Tento způsob vyžaduje vytvoření layoutů pro každý druh zvlášť a může tedy značně navýšit velikost aplikace. Třetím způsobem je použití anotací a *Jungles*. Tento způsob umožňuje označit části kódu, které se zkompilují pouze na určené

zařízení a ostatní ignorují. Nevýhodou řešení je vysoký nárust počtu řádků kódu, ale zkompilovaná aplikace obsahuje pouze kód určený pro dané zařízení.

```
(:round218)
function onUpdate(dc) {
  //implementace funkce
}

(:rectangle240)
function onUpdate(dc) {
  //implementace funkce
}
```

Výpis kódu 5.1: Příklad použitých anotací

Na výpisu 5.1 je příklad použitých anotací, které rozlišují mezí kulatým displejem o velikosti 218x218 pixelů a hranatým displejem o velikosti 240x240 pixelů. Po označení částí kódu je nutné anotace zapsat do *monkey.jungle* souboru, aby při kompilaci došlo k jejich použití.

```
venusq.excludeAnnotations = round218
vivoactive4s.excludeAnnotations = rectangle240
```

Výpis kódu 5.2: Zaznačení anotací do *Jungle* souboru

Obrázek 5.1: Zobrazení úvodního pohledu na sporttesteru(zleva vívoactive 4[40], fénix 6 Pro[41], venu Sq[42], MARQ Captain[43], vívoactive 4S[44]

5.3 Algoritmus navigace

První podmínkou pro spuštění navigace je GPS signál, který celou navigaci spouští při získání první informace o aktuální pozici. Druhou podmínkou je obdržení a zpracování všech tří částí navigačních dat. Pokud jsou obě podmínky splněny může začít navigace.

Prvně se kontroluje, zda se jedná o první pozici, pokud ano, je nutné nastavit vzdálenost aktuálního kroku a vykreslit pozici na druhém pohledu. Nejedná-li se o první pozici, dojde k vypočtení uražené vzdálenosti mezi aktuální pozicí a poslední známou pozicí. Tato hodnota se připočte k celkové vzdálenosti a odečte od aktuální vzdálenosti instrukce. Poté se kontroluje, jestli není aktuální vzdálenost menší jak 4 metry. Když je vzdálenost menší jak 4 metry dochází k přechodu na další instrukci v pořadí. Jako poslední se nasatvuje obrázek znázorňující aktuální instrukci a text instrukce. Diagram algoritmu lze nalézt v příloze A.

5.3.1 Haversine metoda

K výpočtu vzdálenosti dvou bodů algoritmus využívá haversine metodu[45]. Jedná se o vzorec, který vypočítá ortodromu mezi dvěma body na kouli vzhledem k jejich zeměpisné šířce a délce podle vzorce $hav(\theta) = hav(\varphi_1 - \varphi_2) + \cos(\varphi_1)\cos(\varphi_2)hav(\lambda_1 - \lambda_2)$, kde φ_1 a φ_2 jsou zeměpisné šířky bodů a λ_1 a λ_2 jsou zeměpisné délky bodů.

5.4 Úvodní pohled

Hned po spuštění aplikace uživatel uvidí jako první úvodní pohled. Tento pohled slouží k indikaci v jakém stavu se aplikace aktuálně nachází. Stav je zde znázorněn dvěma ukazateli. První ukazatel ukazuje pomocí teček na obrazovce v jaké fázi je přenos dat. Každá tečka reprezentuje jednu zprávu jak je ukázáno na 4.8. Cílem je, aby uživatel věděl v jaké fázi se zrovna nachází a zda proces zpracování instrukcí začal. Druhý ukazatel uživatele informuje o stavu GPS signálu. Slovně je popsán ve spodní části pohledu.

Pokud jsou data navigace úspěšně zpracována(na obrazovce jsou tři černé tečky) a je k dispozici GPS signál, zobrazí se u tlačítka ENTER zelené kolečko. To signalizuje, že je možné stiskem tlačítka zapnout navigaci a přepnout se do druhého pohledu.

5.4.1 Objekt pohledu a ovládání

O vykreslení se stará třída *LoadView*, která dědí z *WatchUi.View*. Jejím úkolem je tedy implementovat metody *initialize, onLayout, onShow, onUpdate* a *onHide*. Z důvodu implementace pro více modelů je zde metoda *onUpdate* celkem 5x. Liší se jen v konstantních hodnotách, které udávají pozice jednotlivých prvků na displeji. Metoda jako první vyčistí celý displej a až potom vykresluje jednotlivé informace.

Vstup od uživatele zpracovává třída *LoadViewBehaviorDelegate*. Ta dědí z *WatchUi.BehaviorDelegate*. Výhodou *BehaviorDelegate* je, že narozdíl od *InputDelegate*, který operuje na úrovni jednotlivých kódů vstupu, tak *BehaviorDelegate* obaluje tyto vstupy do jednotných akcí. Například chování přechodu mezi stránkami je na dotykových zařízeních mapováno na potáhnutí nahorů a dolů, kdežto na klasických zařízeních na tlačítka. *BehaviorDelegate* abstrahuje toto chování do jednoho a je tedy schopen zpracovat oba druhy.

Obrázek 5.2: Úvodní pohled signalizující status aplikace

5.5 Pohled s instrukcemi

Po zapnutí navigace na úvodním pohledu je uživateli ukázán pohled s instrukcemi. Mezi tímto a pohledem s trasou lze volně přepínat. Ovládání se liší dle modelu sporttesteru. Na tomto pohledu uživatel vidí obrázek aktuální instrukce, textový popis, zbývající vzdálenost do další instrukce a celkovou uraženou vzdálnost. Tyto údaje se vypočítavají v hlavním algoritmu navigace a pohled má na starost pouze jejich vykreslení.

5.5.1 Načtení obrázku s instrukcí

Obrázky instrukcí se načítají v hlavním objektu aplikace v navigačním algoritmu. Tyto obrázky musí být zapsány do soubory *drawables.xml*, protože Monkey C kompilátor tento soubor kompiluje do databáze zdrojů, odkud je pak za běhu načítá. Načtení takového zdroje může být velmi drahé z pohledu výkonu, a proto by k němu nemělo docházet při překreslování displeje.

```
<drawables>
    <bitmap id="LauncherIcon" filename="launcher_icon.png"/>
    <bitmap id="right" filename="./directions/direction_continue_right.png"/>
    <bitmap id="slightRight" filename="./directions/
        direction_continue_slight_right.png"/>
</drawables>
```

Výpis kódu 5.3: XML soubor s obrázky

Obrázek 5.3: Pohled s aktuální instrukcí

5.5.2 Ovládání dle typu zařízení

Přechody mezi pohledy a ovládání pohledu s trasou je nutné implementovat na sporttesterech rozlišně dle typu zařízení. Zařízení s dotykovým displejem disponují mnohem více možnostmi, jak lze získat vstup od uživatele. Tlačítkové zařízení jsou omezeny pouze na tlačítka, které nejsou na všech zařízeních stejná. Proto jsem se rozhodl podporovat tlačítková zařízení, která obsahují minimálně tlačítka UP, DOWN a ENTER.

Tyto tří tlačítka stačí k implementaci přechodu mezi pohledy, pohybu s trasou a přiblížení nebo oddálení. UP a DOWN mají vždy funkci předchozí/další a podle aktuálního módu se rozhoduje dále.

- Mód 1 - přechod mezi pohledy
- Mód 2 - posun trasy doleva a doprava
- Mód 3 - posun trasy nahoru a dolů

- Mód 4 - přiblížení a oddálení trasy

Rozdílné druhy ovládání jsou implementovány pomocí anotací jak je popsáno v kapitole 5.2.

5.6 Pohled s trasou

Třetím pohledem je pohled s obrysem trasy a aktuální polohou uživatele. Pohled má základní funkcionalitu jako přesun trasy po displeji a přiblížení, nebo oddálení. Jednou z překážek při implementaci tohoto pohledu byl přepočet zeměpisných souřadnic na souřadnice displeje.

Obrázek 5.4: Pohled s trasou

5.6.1 Přepočet zeměpisných souřadnic

Při zobrazování souřadnic dochází k převodu z Mercatorova zobrazení(EPSG:3857)[46] do Univerzálního transverzálního Mercatorova systému(UTM)[47]. Po převodu je nutné provést projekci z UTM do 2D systému displeje. Při tomto převodu dochází k mírnému zkreslení.

Kapitola 6

Závěr

Cílem práce bylo seznámení se s Connect IQ platformou a jazykem Monkey C, který slouží k vývoji na tuto platformu a multiplatformním nástrojem Flutter. Praktická část se zabývala návrhem a implementací aplikací na obě zmíněné platformy.

Na začátku teoretické části popisuji možnosti vývoje aplikací na Connect IQ platformu, způsob publikování aplikací do Connect IQ Store a Connect IQ API, které se schovává pod názvem Toybox. Dozvěděl jsem se, že ne všechny aplikace na sporttestery disponují stejnou funkcionalitou a často může být jejich výkon omezen výměnou za delší výdrž baterie. Při popisu a práci s jazykem Monkey C jsem ocenil jeho inspiraci ostatními jazyky, protože práce v něm je často velmi intuitivní a uplatnil jsem zde znalosti z jiných jazyků. Nástroj Flutter naopak byl pro mě něčím novým. Dart, jazyk ve kterém je Flutter vytvořen se stejně jako Monkey C inspiroval ostatními jazyky a díky lepší podpoře vývojových prostředí je s ním práce velmi pohodlná.

V praktické části jsou popsány řešení obou aplikací. První je popsána mobilní aplikace, jsou zde vysvětleny možnosti state managementu a ukázka implementace v aplikaci. Dále se kapitola zabývá popisem a návrhem cloudové části aplikace Firebase, která se stará o autorizaci a uložiště dat. Práce s Firebase je velmi přívětivá pro vývojáře a poprvé jsem se setkal s NoSQL databází. Kapitola pokračuje popisem řešení jednotlivých funkcí a komunikací s Garmin zařízením. Až na práci s *flutter_map* balíčkem byla implementace bez překážek. První překážky vznikaly až při implementaci CIQ aplikace a zajištění komunikace mezi oběma aplikacema. Velká překážka je zde nemožnost přímé komunikace s Garmin zařízením, ale nutnost veškerou komunikaci provádět skrze jejich Connect IQ aplikaci a SDK, které poskytují. Zmiňované SDK nefunguje dobře a komunikace se občas zasekává skrz jejich Connect IQ. Tohle představuje komplikaci i z hlediska Flutteru, protože je nutné psát platform-specific kód a provádět komunikaci přes nativní kanál. Kapitola zabývající se popisem CIQ aplikace osvětluje návrh a funkčnost aplikace. V kapitole je vysvětleno jakým způsobem je řešena podpora více druhů zařízení a jak funguje algoritmus navigace. Druhá část kapitoly popisuje jednotlivé pohledy aplikace a jejich funkcionalitu. Omezujícím faktorem pro vývoj složitějších aplikací je omezená výkonnost zařízení. Upřednostňuje se delší výdrž baterie před výkonem

a podle toho se musí řídit i aplikace. Je tedy nutné optimalizovat kód a často je upřednostňován nekvalitní kód výměnou za lepší výkon, nebo menší nároky na paměť.

Výsledkem práce jsou dvě spolupracující aplikace. Jedna na mobilní telefony a druhá pro mnoho druhů Garmin sporttesterů. Cílem práce bylo otestovat nástroj Flutter pro vývoj aplikací s pokročilejšími funkcemi jako jsou např. mapy a to se povedlo. Přívětivost *flutter_map* dokumentace není dobrá a momentálně není jiná možnost jak do aplikací vložit open-source mapy. Do budoucna by bylo aplikaci vhodné rozšířit o určitou formu uživatelských profilů. Momentální řešení nabízí jen autorizaci pomocí účtů, ale uživatel už např. nevidí své statistiky, přehled používaných zařízení apod. Aplikace na sporttestery přináší možnost navigace i na nižších modelech, které tuto funkcionalitu nenabízí. V aktuální verzi aplikace neumožňuje zaznamenávat aktivitu při zapnuté navigace. Vzhledem k tomu, že Garmin sporttestery jsou silně orientované na sportovce, bylo by vhodné tuto funkci dodělat. Vypracováním jsem získal mnoho zkušeností, které mě posunuly zase o kus dále v oblasti vývoje mobilních aplikací.

Literatura

1. *Connect IQ SDK* [online]. 2021 [cit. 2021-04-06]. Dostupné z: `https://developer.garmin.com/connect-iq/overview/`.
2. *Garmin announces ability to develop apps on wearables, with Connect IQ* [online]. 2014 [cit. 2021-04-06]. Dostupné z: `https://www.dcrainmaker.com/2014/09/garmin-connect-iq-apps.html`.
3. *Toybox documentation* [online]. 2021 [cit. 2021-04-06]. Dostupné z: `https://developer.garmin.com/connect-iq/api-docs/index.html`.
4. *Bluetooth* [online]. 2021 [cit. 2021-04-06]. Dostupné z: `https://www.bluetooth.com/`.
5. *Connect IQ API Support* [online]. c2021 [cit. 2021-04-06]. Dostupné z: `https://developer.garmin.com/connect-iq/monkey-c/objects-and-memory/`.
6. *Monkey C instanceof and has* [online]. c2021 [cit. 2021-04-06]. Dostupné z: `https://developer.garmin.com/connect-iq/monkey-c/functions/`.
7. KERNIGHAN, Brian W.; RITCHIE, Dennis M. *Programovací jazyk C*. 2. vydání. Brno: Computer Press, 2019. ISBN 978-80-251-4965-2.
8. SARANG, Poornachandra G. *Java programming*. New York: McGraw-Hill, c2012. ISBN 978-0-07-163360-4.
9. POWELL, Thomas A.; SCHNEIDER, Fritz. *JavaScript: the complete reference*. 3rd ed. New York: McGraw-Hill, c2012. ISBN 978-0-07-174120-0.
10. LUTZ, Mark. *Learning Python*. Vydání 5. O'Reilly Media, 16.7.2013. ISBN 978-1449355739.
11. LERUSALIMSCHY, Roberto. *Programming in Lua*. Vydání 4. Lua.Org, 1.8.2016. ISBN 8590379868.
12. THOMAS, Dave; HUNT, Andy; FOWLER, Chad. *Programming Ruby: The Pragmatic Programmers' Guide*. Vydání 4. Pragmatic Bookshelf, 4.7.2013. ISBN 978-1937785499.
13. LERDORF, Rasmus. *PHP4: kapesní příručka*. Gliwice: Helion, c2004. ISBN 83-7361-463-X.
14. *Duck Typing* [online]. 2020 [cit. 2021-04-06]. Dostupné z: `https://devopedia.org/duck-typing`.

15. STROUSTRUP, Bjarne. *C++ programovací jazyk*. 1. české vyd. Praha: BEN - technická literatura, 1997. ISBN 80-901507-2-1.

16. BRACHA, Gilad. *The Dart Programming Language*. Vydání 1. Addison-Wesley Professional, 2015. ISBN 0321927702.

17. *Companies using Flutter* [online]. 2021 [cit. 2021-04-06]. Dostupné z: `https://flutter.dev/showcase`.

18. *Dart type system* [online]. 2021 [cit. 2021-04-06]. Dostupné z: `https://dart.dev/guides/language/type-system`.

19. *Dart overview* [online]. 2021 [cit. 2021-04-06]. Dostupné z: `https://dart.dev/overview`.

20. SCHWEIGER, Frederik. The Layer Cake: How Flutter uses Widgets, Elements and RenderObjects to create delicious eye-candy at 120fps. *Medium* [online]. [B.r.] [cit. 2021-04-06]. Dostupné z: `https://medium.com/flutter-community/the-layer-cake-widgets-elements-renderobjects-7644c3142401`.

21. *Flutter architectural overview* [online]. 2021 [cit. 2021-04-06]. Dostupné z: `https://flutter.dev/docs/resources/architectural-overview`.

22. *Bloc: Dart state management package* [online] [cit. 2021-04-12]. Dostupné z: `https://bloclibrary.dev/`.

23. *Redux* [online] [cit. 2021-04-12]. Dostupné z: `https://pub.dev/packages/redux`.

24. *Riverpod: Provider, but different* [online] [cit. 2021-04-12]. Dostupné z: `https://riverpod.dev/`.

25. *OpenRouteService* [online] [cit. 2021-04-12]. Dostupné z: `https://openrouteservice.org/`.

26. BLAIS, Shawn. *Flutter: WidgetView – A Simple Separation of Layout and Logic* [online]. 2020 [cit. 2021-04-12]. Dostupné z: `https://blog.gskinner.com/archives/2020/02/flutter-widgetview-a-simple-separation-of-layout-and-logic.html`.

27. *Apple ID* [online]. Wikimedia Foundation, 2001-2021 [cit. 2021-04-12].

28. *OAuth 2.0* [online]. 2012 [cit. 2021-04-12]. Dostupné z: `https://oauth.net/2/`.

29. *OpenID Connect* [online]. 2014 [cit. 2021-04-12]. Dostupné z: `https://openid.net/connect/`.

30. *Flutter map package* [online]. 2021 [cit. 2021-04-12]. Dostupné z: `https://pub.dev/packages/flutter_map`.

31. *Leaflet* [online]. 2010-2021 [cit. 2021-04-12]. Dostupné z: `https://leafletjs.com/`.

32. *OpenStreetMap* [online] [cit. 2021-04-12]. Dostupné z: `https://www.openstreetmap.org/`.

33. *OpenTopoMap* [online] [cit. 2021-04-12]. Dostupné z: `https://opentopomap.org/`.

34. *WMFlabs tiles* [online]. 2018-2021 [cit. 2021-04-12]. Dostupné z: https://tiles.wmflabs.org/.

35. *Map controller package* [online] [cit. 2021-04-12]. Dostupné z: https://pub.dev/packages/map_controller.

36. *Pelias Geocoder* [online] [cit. 2021-04-12]. Dostupné z: https://pelias.io/.

37. *Simplify dart* [online]. 2020 [cit. 2021-04-12]. Dostupné z: https://pub.dev/packages/simplify_dart.

38. *Ramer-Douglas-Pecker algorithm* [online]. Wikimedia Foundation, 2020 [cit. 2021-04-12]. Dostupné z: https://en.wikipedia.org/wiki/Ramer%5C%E2%5C%80%5C%93Douglas%5C%E2%5C%80%5C%93Peucker_algorithm.

39. MOKRZYCKI, Wojciech; SAMKO, M. New version of Canny edge detection algorithm. 2012.

40. *Garmin vívoactive4* [online] [cit. 2021-04-20]. Dostupné z: https://www.garmin.cz/garmin-vivoactive4-silvergray-band/80206.

41. *Garmin fenix6 PRO* [online] [cit. 2021-04-20]. Dostupné z: https://www.garmin.cz/garmin-fenix6-pro-glass-blackblack-band-mapmu/80371.

42. *Garmin Venu Sq* [online] [cit. 2021-04-20]. Dostupné z: https://www.garmin.cz/garmin-venu-sq-music-slateblack-band/81263.

43. *Garmin MARQ Captain* [online] [cit. 2021-04-20]. Dostupné z: https://www.garmin.cz/marq/captain.

44. *Garmin vívoactive® 4S* [online] [cit. 2021-04-20]. Dostupné z: https://www.garmin.cz/garmin-vivoactive4s-silvergray-band/80202.

45. *Haversine formula* [online]. Wikimedia Foundation, 2001-2021 [cit. 2021-04-20]. Dostupné z: https://en.wikipedia.org/wiki/Haversine_formula.

46. *EPSG:3857* [online]. 25.11.2015 [cit. 2021-04-20]. Dostupné z: https://epsg.io/3857.

47. *Universal Transverse Mercator coordinate system* [online]. Wikimedia Foundation, 2001-2021 [cit. 2021-04-20]. Dostupné z: https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system.

Příloha A

Diagram navigačního algoritmu